

The use of Multi-threaded Java, JDBC and On-the-fly Java-Generated SPL code for Automatic Application Development

Zachi Klopman

Cambridge Interactive Development Corp.

Change Session Code

Day, April 0, 2008 • 00:00 a.m. – 00:00 a.m.

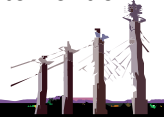
2008 IIUG Inform*i*x Conference



The Problem

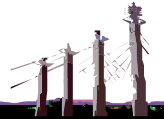
- Typical large databases contain thousands of tables
- Tables may be added and table schema may be changed on a daily basis
- Applications require synchronization of hundreds of tables (or views), whose schemas may also change frequently
- Mix of ANSI and Informix mode (as well as other databases and 3rd party service providers) prevents direct inter-database transactions

We need a self-adapting system with minimal human intervention



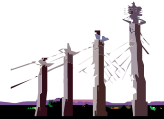
Solution: JDBC Query Metadata

- JDBC has a well defined interface for analyzing SQL output: ResultSetMetaData
- Classes that implement this interface provide access to the details of the various fields that are returned as output to a query (the ResultSet)
 - The actual classes usually depend on the JDBC driver



ResultSetMetaData Basic Methods

- `getColumnCount()` – returns number of columns
- `getColumnType(col #)` – returns JDBC type code
- `getColumnTypeName(col #)` – returns type name
- `columnName(col #)` – returns name of column
- `isNullable(col #)` – can the column accept NULL?
- `getColumnDisplaySize(col #)` – size of column



Running Example

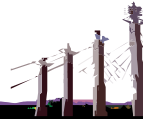
```
CREATE TABLE tab1 (  
    col1 INT8 NOT NULL,  
    col2 INT,  
    col3 VARCHAR(40),  
    col4 DECIMAL(10,2)  
);  
CREATE UNIQUE INDEX i1 ON tab1(col1);  
CREATE INDEX i2 ON tab1(col2,col3);
```



ResultSetMetaData Example I

JDBC Query Code:

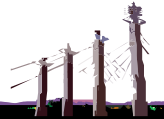
```
Connection con = ...;  
Statement st = con.createStatement();  
ResultSet rs = st.executeQuery(  
    "SELECT *,col1*col2 AS ex1, col1+1 FROM tab1");  
ResultSetMetaData md = rs.getMetaData();
```



ResultSetMetaData Example II

Some values from the MetaData returned:

- `md.getColumnCount()` → 5
- `md.getColumnName(1)` → `coll`
- `md.getColumnName(5)` → `ex1`
- `md.getColumnName(6)` → `(expression)`
- `md.getColumnType(2)` → 4 *java.sql.Types.INTEGER*
- `md.getColumnType(3)` → 12 *java.sql.Types.VARCHAR*
- `md.getColumnDisplaySize(3)` → 40
- `md.getColumnType(4)` → 3 *java.sql.Types.DECIMAL*
- `md.getPrecision(4)` → 10
- `md.getScale(4)` → 2



Using MetaData Example I

Goal:

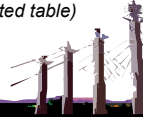
- Generate an SQL statement that returns all identical rows in a table

Solution:

- loop on the metadata to generate this output:

```
SELECT UNIQUE a.*  
FROM tab1 a, tab1 b  
WHERE a.col1=b.col1 AND a.col2=b.col2 AND  
      a.col3=b.col3 AND a.col4=b.col4 AND  
      a.rowid<>b.rowid
```

(Assuming non-fragmented table)

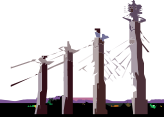


Using MetaData Example II

This is the code that generates the SQL:

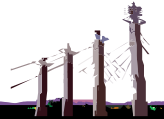
```
ResultSet rs = st.executeQuery("SELECT * FROM "+tab);
ResultSetMetaData md = rs.getMetaData();
String sql = "SELECT UNIQUE a.* "+
    "FROM "+tab+" a, "+tab+" b "+ "WHERE ";
for (int i=1;i<=md.getColumnCount();i++) {
    String c = md.getColumnName(i);
    sql += (c+"<>" + c + " AND ");
}
sql += "a.rowid<>b.rowid";
```

Column counts starts at 1, just to annoy us.



Prepared Statements

- Using the knowledge of the tables, one can also automate the creation of prepared SQL statements
- Using prepared statements saves significant time vs. creating and parsing a new statement for every record
 - It also protects from “SQL-Injection”
- The statements will use order-based variables
 - Some JDBC drivers (such as the new Informix driver) support call-by-name variables, which are not very useful for this type of automation



Prepared statements also eliminate “SQL Injection Hack”, but that is not relevant for this presentation.

Handling Data Values

- Since we are receiving data values of varying data types, we need to check the returned type and handle each value accordingly.
- Here is a code snippet that handles that:

```
import java.sql.Types;
type = md.getColumnType(i);

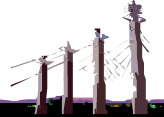
if (type == CHAR || type == VARCHAR
    ||
    type == LONGVARCHAR) {
    ... rs.getString(i);
} else if (type == DECIMAL) {
    ... rs.getBigDecimal(i);
} else if (type == DATE) {
    ... rs.getDate(i);
} else if (type == TIME) {
    ... rs.getTime(i);
} else if (type == TIMESTAMP) {
    ... rs.getTimestamp(i);
} else if (type == BOOLEAN) {
    ... rs.getBoolean(i);
}
```

And so on... or so it seems. But it is not that easy...



Primitive Data Types and NULLS

- While Java handles NULL values for objects (such as strings and Decimals) nicely, it fails to do so for primitive data types (such as int, int8 and double)
- There is a `wasNull()` method, to find if last value used was null (as well as `setNull()` to set null values), but these are cumbersome to use.
- In particular, `wasNull()` only applies to last read parameter – not to a column number (or name)

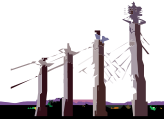


Primitive Data Types and NULLS

- So in order to do primitive values, one needs a piece of code that resembles this:

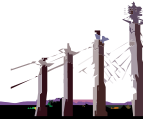
```
type = md.getColumnType(j);  
  
if (type==java.sql.Types.INTEGER) {  
    int i = s.getInt(j);  
    if (s.isNull()) {  
        // value was null  
        ...  
    } else {  
        // i is a valid value  
        ... i ...  
    }  
}
```

- To make things easier, one needs many convenience methods for this type of work.



Informix-Specific Types and Mapping

- There are some data types, which are platform-specific. Informix, for example, has SETs, MULTISets, ROWs, SBLOBs and more.
- ROWs are mapped to `java.sql.Struct`
- SETs and MULTISets are mapped to `java.sql.Array`
 - Informix also optionally maps SETs to `HashSet` and MULTISets to `TreeSet`
- In order to get values, one may use these methods:
 - `generic getObject`
 - `getString` for String representation
 - Informix-specific methods



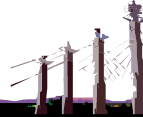
Informix SET Handling Code Sample

```
// read the set from the ResultSet object
HashSet hs = (HashSet) rs.getObject(i);

// do something with the set values
for (Object o: hs) {
    // o is an iterator for the set
}
// maybe add some new value(s) to the set
hs.add(...);

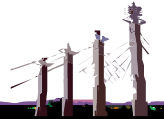
// attach the value into a prepared statement
writeStmt.setXXX(...);
writeStmt.setObject(i, hs);

// write (or update) the new record
writeStmt.execute();
```



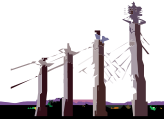
Database Information

- The information about the database and objects within it (tables, procedures, etc.) are accessed via DatabaseMetaData class
- Information includes SQL operations supported, keywords, permissions, behaviors, system functions supported and much more
- For Informix, catalog is the database names, while the schema is user name.
- `DatabaseMetaData.getCatalogs()` will get a list of all databases in an instance



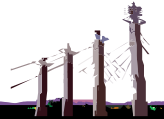
Getting Table Information

- Use the `getTables()` method to access the list of tables
- One can choose table types, which also includes system tables, views and synonyms (as well as other non-Informix table types).
- One can specify a table name pattern
- `getSuperTables()` is used for hierarchal tables



Identifying Primary (and Alt) Keys

- To find the primary key, use `getPrimaryKeys()`
- If a Primary key is not defined, look for unique indices, as one can be an alternate key
 - Sometimes a multi-column Primary key is not easy to use – another reason to look for an alternate key
 - Sometimes an alternate key will allow nulls. These are not useful (unless there are no nulls present).
- If there is more than a single unique index, human intervention is required to identify best key to use.



The DBA's Dilemma

- Java is easy to write, but slow due to having only TCP connections between the Virtual Machine and DB and requirements of the VM itself (memory & cpu).
- We could run a Java VM within the instance, but:
 - Never use anything that isn't fully tested
 - Apps change constantly...and we have hundreds of them!
 - Never use anything that can cause major problems
 - Running arbitrary Java code in (or on) the server? Scary...
 - Java in the server is several generations behind Sun
 - No generics, for example

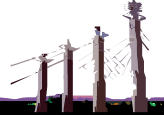
Remember: DBAs are Extremely Paranoid!



Solution: Generate SPL

- SPL's have been used since Online 4 – extremely reliable
- Limited in power and possibility to cause engine faults
 - *But they still do...unfortunately**.
- They are very efficient
- Run in the engine - very little data transfer is needed
- Since they are generated, there is no need to test each one individually – only the generation procedure and samples

**we have an open PMR on this*



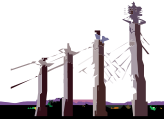
Performance: SPL vs. Java/JDBC

- Test Program 1 – the least possible work for Java
 - generate and insert 40M records to tab1 (no reading data)
 - commit every 800K records
 - Java uses batch and PUT inserts for max efficiency
- Environment:
 - Java 5: Sun Linux 64 bit
 - Communications with IDS via sockets (loopback)
 - IDS 11.1 FC2, 25GB memory, buffered logging
 - Informix JDBC driver 3.0
- Results (Java vs. SPL):
 - Indices enabled: 1653s vs. 1589
 - Indices disabled (less server work): 960 vs. 996
 - These may seem about the same, but Java utilizes two processors!



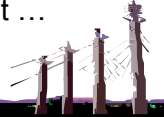
Performance: SPL vs. Java/JDBC

- Test Program 1 – more work for Java
 - select 10M records from an existing table to tab1 (with minor data changes, all in the `select` statement and therefore in server)
 - commit every 100K records
 - Java uses batch and PUT inserts for max efficiency
- Environment:
 - Java 5: Sun Linux 64 bit
 - Communications with IDS via sockets (loopback)
 - IDS 11.1 FC2, 25GB memory, buffered logging
 - Informix JDBC driver 3.0
- Results (Java vs. SPL):
 - Indices disabled (less server work): 260 vs. 231
 - SPL is faster when data needs to be moved



SPL Code Generation Procedure

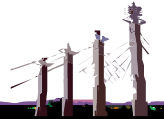
- Run the SELECT to get the appropriate metadata
- Loop over the various fields returned, generating the field lists and the various loop elements
 - Examples of loop elements:
 - `DEFINE col1_var LIKE tab1.col1;`
 - Examples of lists:
 - `col1,col2,...`
 - `col1=col1 AND col2=col2 AND ...`
 - `?,?,?,?` (placeholders for PREPARE statements)
- Assemble the pieces (preferably in `StringBuffer`, but it does not really matter performance-wise) and we get ...



Code Sample for Creating Lists and Loop Elements

```
StringBuffer sp = new StringBuffer(10000);
boolean isNotFirst = false;
sp.append("CREATE PROCEDURE proc_"+tableName+"();\n");
sp.append("  DEFINE i INT;\n  DEFINE j INT;\n");
for (int i = 1; i <= md.getColumnCount(); i++) {
    if (isNotFirst) {
        q1.append(",");
        q2.append(",");
    } else {
        isNotFirst = true;
    }
    q1.append(md.getColumnName(i));
    q2.append("?");

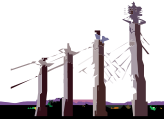
    sp.append("  DEFINE ");
    sp.append(md.getColumnName(i)+" ");
    sp.append(md.getColumnTypeName(i)+";\n");
}
```



Case Study: Synchronization with External Email Service Provider

Requirements:

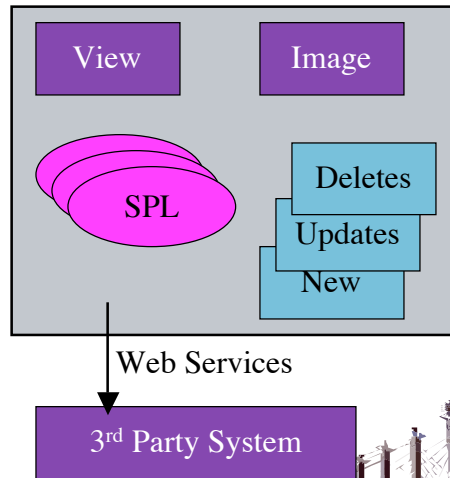
- Dozens of tables and views which need to be synchronized with 3rd party via Web Services interface – which could change on a whim
- As little human involvement as possible to reduce work and possibility of errors
- Due to size of data sources, only transfer changes
- Since this is a 3rd party, we must prepare for any fault on their side (usual paranoia...)
 - Therefore must keep an image of their data sets



Solution Architecture

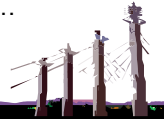
Automatically create:

- Image and Temporary change tables
- SPL code to identify changes (new, update, del)
- SPL code to apply changes to image
- Web services code to create data structures on 3rd party and apply changes to them



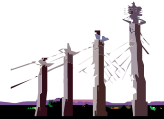
Multithread Issues I

- Thread priority and scheduling
 - A higher-priority thread may run and not let a lower-priority thread run
 - Just having many threads running simultaneously can hold up the SQL operations
 - This may cause a transaction to abort due to timeouts
- Locking Issues
 - Java, as well as Informix, locks objects – another source for deadlocks.
 - These, however, will not be detected by the server.
 - But, eventually, the database will timeout and abort the transaction, which will clear the deadlock.
 - And the application has to handle the failed transaction...



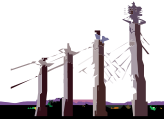
Multithread Issues II

- Exceptions
 - Original Java threads (`Runnable`) do not return values or throw exceptions
 - Therefore, SQL Exceptions need to be taken care of in the thread itself, not the calling thread or main application
 - Java 5 added exception handler for threads, as well as the `Callable` interface, which remedies these issues.



Summary

- Java/JDBC, although effective for database applications, suffers from communications latency and bandwidth (and other performance issues)
- Running Java/JDBC inside the server introduces limitations and performance issues, as well as unacceptable risks to critical production systems (***DBAs are always paranoid***)
- Stored Procedures (written in Informix Stored Procedure Language – SPLs) are efficient and (relatively) safe
 - as long as no goto is used...(new Cheetah SPL feature)
- Java/JDBC may be used to generate SPLs to safely alleviate performance issues



Session #####

Session Title

Zachi Klopman

Cambridge Interactive Development Corp.

zklopman@cidc.com

